In [6]:

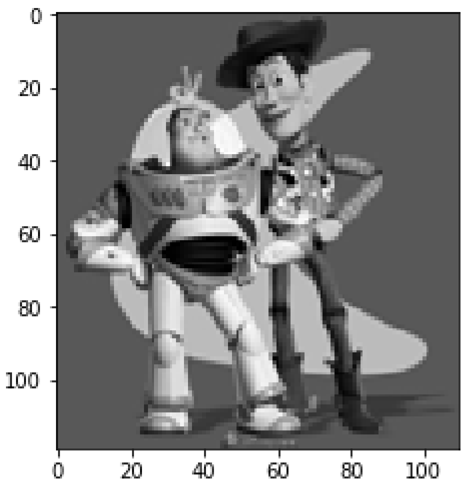
```
import cv2
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from utils import *
import os
#from scipy.sparse import csr_matrix
from scipy.sparse.linalg import lsqr
from scipy.sparse import csr_matrix
```

In [91]:

```
toy_img = cv2.cvtColor(cv2.imread('samples/toy_problem.png'), cv2.COLOR_BG
R2RGB)
toy_img = cv2.cvtColor(toy_img, cv2.COLOR_BGR2GRAY).astype('double') / 255
.0
plt.imshow(toy_img, cmap="gray")
```

Out[91]:

```
<matplotlib.image.AxesImage at 0x21046f44780>
```



In [92]:

```
toy_img.shape
```

Out[92]:

```
(119, 110)
```

Part 1 Toy Problem (20 pts)

In [105]:

```
def toy_reconstruct(toy_img):
    """
    The implementation for gradient domain processing is not complicated,
but it is easy to make a mistake, so let's start with a toy example. Recon
struct this image from its gradient values, plus one pixel intensity. Deno
te the intensity of the source image at (x, y) as s(x,y) and the value to
solve for as v(x,y). For each pixel, then, we have two objectives:
    1. minimize (v(x+1,y)-v(x,y) - (s(x+1,y)-s(x,y)))^2
    2. minimize (v(x,y+1)-v(x,y) - (s(x,y+1)-s(x,y)))^2
    Note that these could be solved while adding any constant value to v,
so we will add one more objective:
    3. minimize (v(1,1)-s(1,1))^2
    :param toy_img: numpy.ndarray
    """
    rows, cols = toy_img.shape

    im2var = np.arange(rows * cols).reshape(rows, cols)
    print(im2var)

    size_toy = toy_img.size


    equations_num = 2 * size_toy + 1
    print(equations_num, size_toy)
    A = np.zeros(shape = (equations_num, size_toy))
    b = np.zeros(shape = (equations_num, 1))
    e = 0



    for y in range(0,rows):
        for x in range(0, cols - 1):
            A[e][im2var[y][x+1]] = 1
            A[e][im2var[y][x]] = -1
            b[e] = toy_img[y][x+1] - toy_img[y][x]
            e = e + 1




    for y in range(0,rows -1):
        for x in range(0, cols):
            A[e][im2var[y+1][x]] = 1
            A[e][im2var[y][x]] = -1
            b[e] = toy_img[y+1][x] - toy_img[y][x]
            e = e + 1




    A[e][im2var[0][0]] = 1
    b[e] = toy_img[0][0]
    print("start")
    print(A.shape)
    print(b.shape)
    v = lsqr(A, b)
    print("done")

    im_out = np.reshape(v[0], (rows, cols))

    return im_out
```

In [106]:
```
im_out = toy_reconstruct(toy_img)
```

```
[[    0     1     2 ...,   107   108   109]
 [  110   111   112 ...,   217   218   219]
 [  220   221   222 ...,   327   328   329]
 ...,
 [12760 12761 12762 ..., 12867 12868 12869]
 [12870 12871 12872 ..., 12977 12978 12979]
 [12980 12981 12982 ..., 13087 13088 13089]]
26181 13090
start
(26181, 13090)
(26181, 1)
done
```

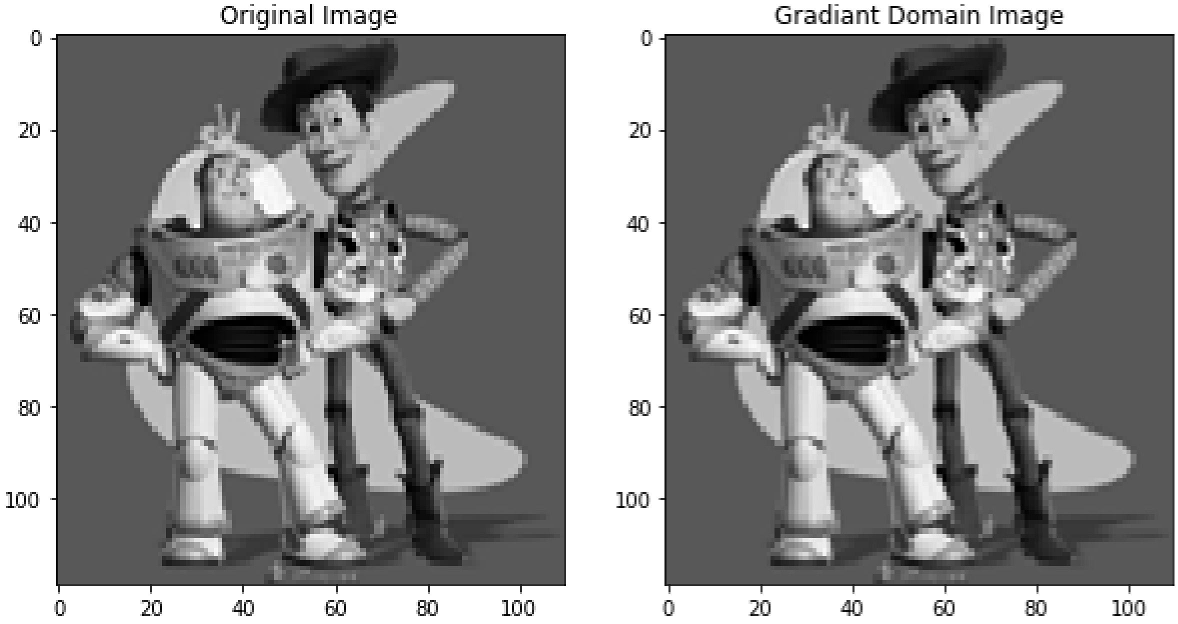
In [95]:
```
im_out
```

Out[95]:
```
array([[ 0.34509988,  0.34510081,  0.34510143, ...,  0.34510012,
         0.34510014,  0.34510012],
       [ 0.34510083,  0.34510115,  0.34510159, ...,  0.34510016,
         0.34510017,  0.34510013],
       [ 0.34510142,  0.34510152,  0.34510176, ...,  0.34510029,
         0.34510029,  0.34510024],
       ...,
       [ 0.34509328,  0.34509327,  0.34509327, ...,  0.34509293,
         0.34509279,  0.34509275],
       [ 0.34509314,  0.34509316,  0.3450932 , ...,  0.34509307,
         0.34509293,  0.34509286],
       [ 0.34509307,  0.34509308,  0.34509315, ...,  0.34509311,
         0.34509299,  0.34509295]])
```

In [96]:
```
if im_out.any():
    print("Error is: ", np.sqrt(((im_out - toy_img)**2).sum()))
```

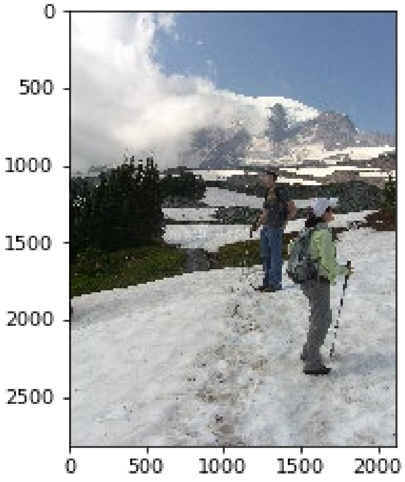
```
Error is:  0.000317018500683
```

In [98]:
```
#Images sanity check
fig, axes = plt.subplots(1, 2)
axes[0].imshow(toy_img,cmap='gray')
axes[1].imshow(im_out,cmap='gray')
axes[0].title.set_text('Original Image')
axes[1].title.set_text('Gradiant Domain Image')
fig.set_size_inches(10, 10)
plt.savefig('toy_problem.png')
```



Preparation

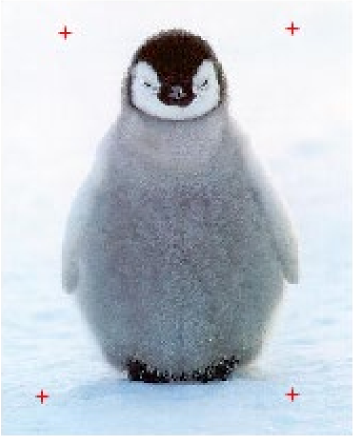
In [190]:
```
# Feel free to change image
background_img = cv2.cvtColor(cv2.imread('samples/im2.JPG'), cv2.COLOR_BGR
2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```



Out[190]:
```
<matplotlib.image.AxesImage at 0x21054468080>
```

In [183]:
```
# Feel free to change image
object_img = cv2.cvtColor(cv2.imread('samples/penguin-chick.jpeg'), cv2.CO
LOR_BGR2RGB).astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

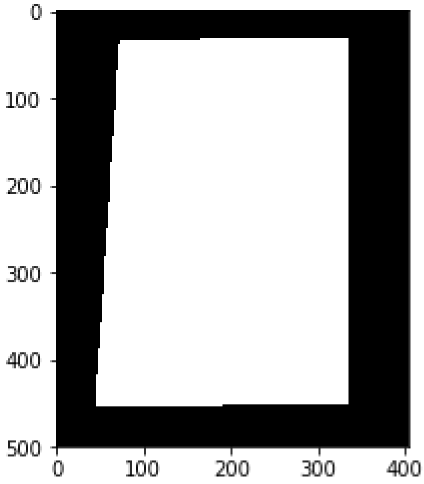
```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [184]:

```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

```
<matplotlib.figure.Figure at 0x210476fa390>
```



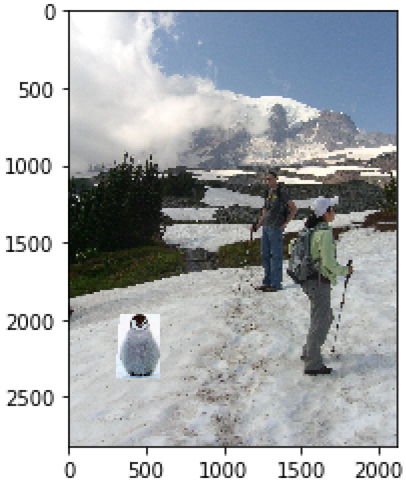
In [194]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
bottom_center = specify_bottom_center(background_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
 Also, make sure the object fill fit into the background image. Otherwise
it will crash
```

In [195]:

```
%matplotlib inline
import matplotlib.pyplot as plt
cropped_object, object_mask = align_source(object_img, mask, background_im
g, bottom_center)
```



Part 2 Poisson Blending (50 pts)

In [89]:

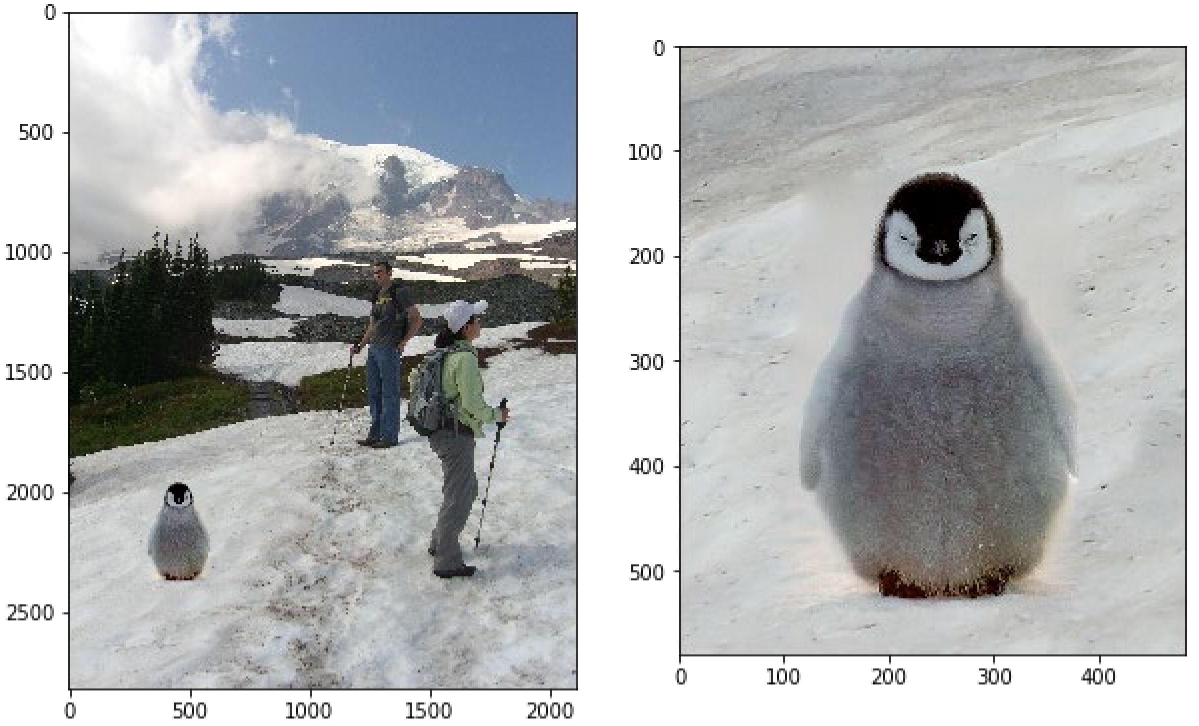
```
def poisson_blend(cropped_object, object_mask, background_img, mask):
    """
    :param cropped_object: numpy.ndarray One you get from align_source
    :param object_mask: numpy.ndarray One you get from align_source
    :param background_img: numpy.ndarray
    """
    #TO DO
    row_start = int(bottom_center[1] - (mask.shape[0]))
    col_start = int(bottom_center[0] - (mask.shape[1]/2))
    pad = 40

    output = background_img
    background_img = background_img[row_start - pad:row_start + mask.shape
[0] + pad, col_start - pad: col_start + mask.shape [1] + pad]
    cropped_object = cropped_object[row_start - pad:row_start + mask.shape
[0] + pad, col_start - pad: col_start + mask.shape [1] + pad]
    object_mask = object_mask[row_start - pad:row_start + mask.shape[0] +
pad, col_start - pad: col_start + mask.shape [1] + pad]
    #return cropped_object





    #return cropped_object, background_img
    background_rows = background_img.shape[0]
    background_cols = background_img.shape[1]
    background_count = background_rows * background_cols

    output_mask = np.zeros(shape = (background_rows, background_cols,3))

    #print(background_count)
    im2var =  np.arange(background_count).reshape(background_rows, backgro
und_cols)

    #print(im2var)


    for z in range(3):
        v = []
        sparse_value = []
        sparse_row = []
        sparse_col = []
        b = []
        e = 0
        #A =  np.zeros(shape=(background_count,background_count))

        for y in range(background_rows):
            for x in range(background_cols):
                if not object_mask[y,x]: #background only
                    sparse_value.append(1)
                    sparse_row.append(e)
                    sparse_col.append(im2var[y,x])
                    b.append(background_img[y,x,z])
                    e = e + 1


                else:
                    if object_mask[y,x+1]:
                        sparse_value.append(-1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x + 1])

                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z]  - cropped_object[y
,x+1,z])
```

```
                        e = e + 1
                    if object_mask[y+1,x]:
                        sparse_value.append(-1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y+1,x])

                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z]  - cropped_object[y+1,x,z])

                        e = e + 1

                    if not object_mask[y,x+1]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z] - cropped_object[y,x+1,z] + background_img[y,x+1,z])

                        e = e + 1

                    if not object_mask[y,x-1]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z] - cropped_object[y,x-1,z] + background_img[y,x-1,z])

                        e = e + 1

                    if not object_mask[y+1,x]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z] - cropped_object[y+1,x,z] + background_img[y+1,x,z])

                        e = e + 1

                    if not object_mask[y-1,x]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        b.append(cropped_object[y,x,z] - cropped_object[y-1,x,z] + background_img[y-1,x,z])

                        e = e + 1

        sparse_value = np.asarray(sparse_value)
        sparse_row = np.asarray(sparse_row)
        sparse_col = np.asarray(sparse_col)
        b = np.asarray(b).T
        #return 0
        A = csr_matrix((sparse_value, (sparse_row, sparse_col)), shape=(e,
 background_count))
        print(z)
        v = lsqr(A, b)

        send = np.clip(v[0], a_min = 0, a_max = 1)
        output[row_start - pad:row_start + mask.shape[0] + pad, col_start
- pad: col_start + mask.shape [1] + pad,z] = np.reshape(send, (background_
rows, background_cols))
        output_mask[:,:,z] = np.reshape(send, (background_rows, background
_cols))


    return output, output_mask
```

In [196]:

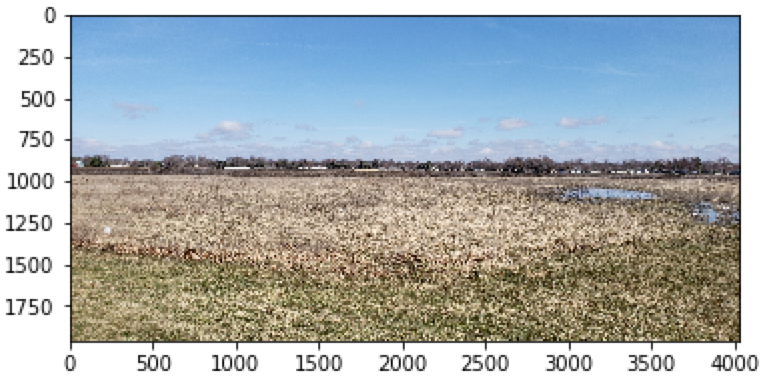
```
output, output_mask = poisson_blend(cropped_object, object_mask, backgroun
d_img,mask)
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(10, 10)
axes[0].imshow(output)
axes[1].imshow(output_mask)
plt.savefig('penguin_mountain.jpg')
```

```
0
1
2
```



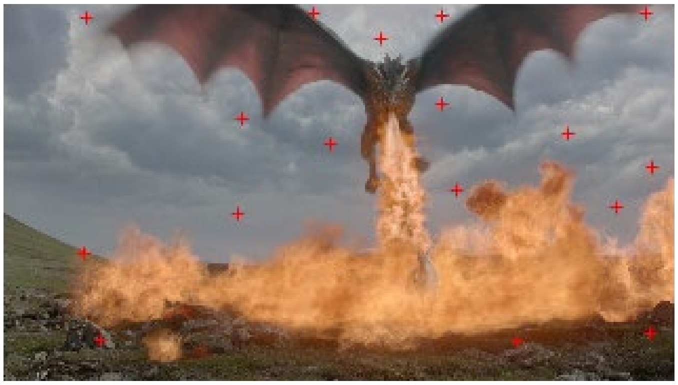
In [143]:
```
background_img = cv2.cvtColor(cv2.imread('samples/feild.JPG'), cv2.COLOR_B
GR2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```

Out[143]:
```
<matplotlib.image.AxesImage at 0x210596df630>
```



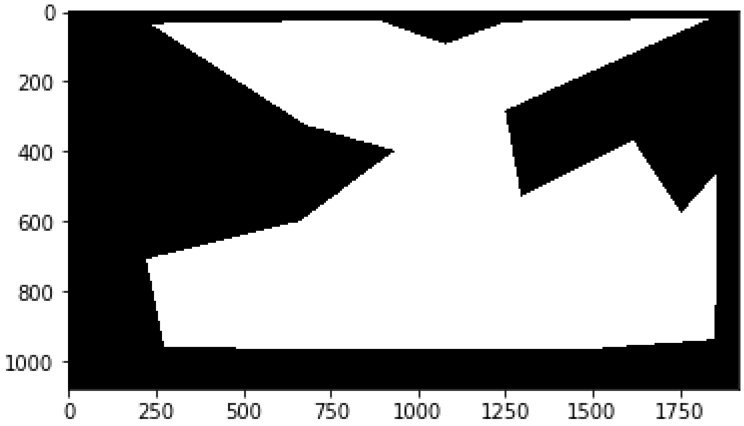
In [138]:
```
# Feel free to change image
object_img = cv2.cvtColor(cv2.imread('samples/dragon_large.jpg'), cv2.COLO
R_BGR2RGB).astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [144]:
```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

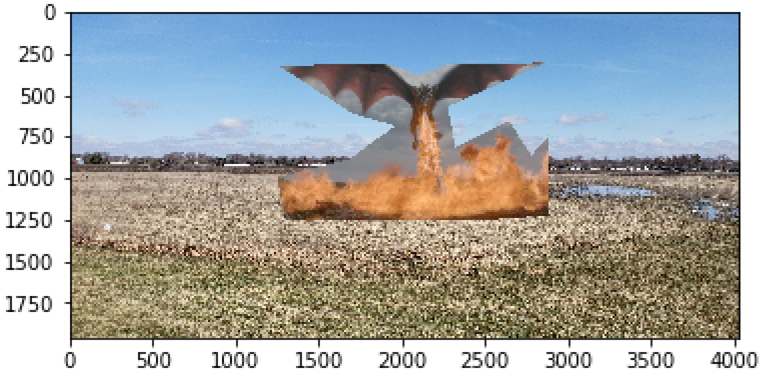
```
<matplotlib.figure.Figure at 0x210489442b0>
```



In [149]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
bottom_center = specify_bottom_center(background_img)
```

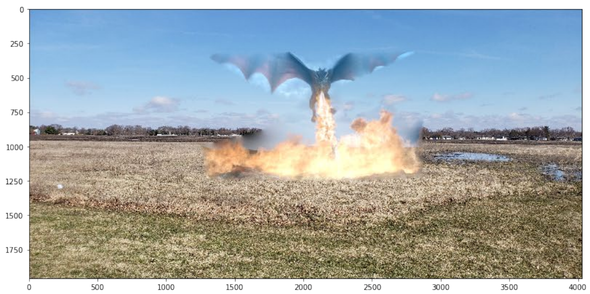
```
If it doesn't get you to the drawing mode, then rerun this function again.
 Also, make sure the object fill fit into the background image. Otherwise
it will crash
```

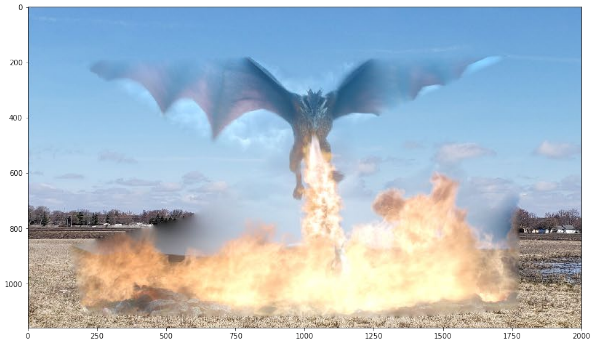
```
In [150]: %matplotlib inline
          import matplotlib.pyplot as plt
          cropped_object, object_mask = align_source(object_img, mask, background_im
          g, bottom_center)
```



```
In [152]: output, output_mask = poisson_blend(cropped_object, object_mask, backgroun
          d_img,mask)
          fig, axes = plt.subplots(1, 2)
          fig.set_size_inches(30, 30)
          axes[0].imshow(output)
          axes[1].imshow(output_mask)
          plt.savefig('dragon_feild.jpg')
```

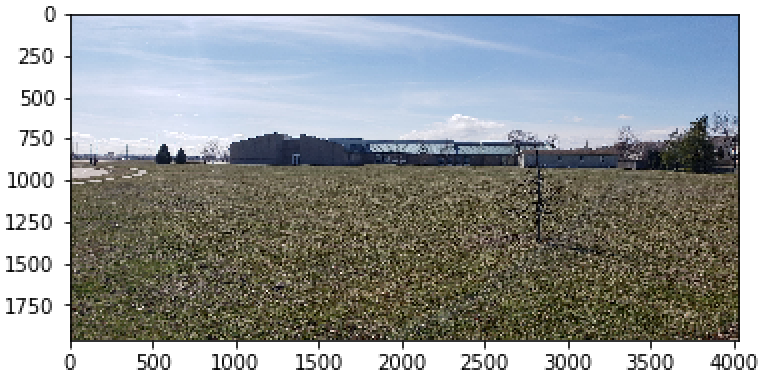
```
0
1
2
```





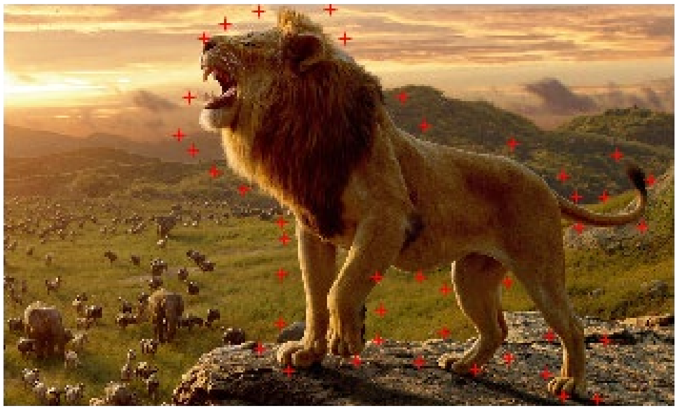
In [157]:
```
background_img = cv2.cvtColor(cv2.imread('samples/barkstall.JPG'), cv2.COLOR_BGR2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```

Out[157]:
```
<matplotlib.image.AxesImage at 0x2104761c048>
```



In [171]:
```
# Feel free to change image
object_img = cv2.cvtColor(cv2.imread('samples/lion.jpg'), cv2.COLOR_BGR2RGB).astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

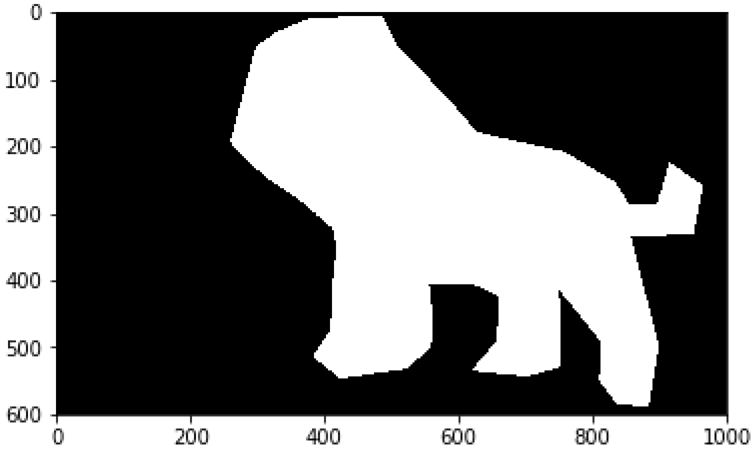
```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [172]:

```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

```
<matplotlib.figure.Figure at 0x210544b04a8>
```



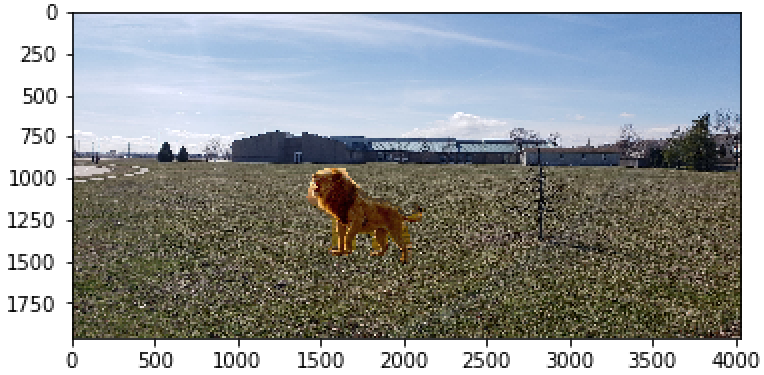
In [175]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
bottom_center = specify_bottom_center(background_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
 Also, make sure the object fill fit into the background image. Otherwise
it will crash
```

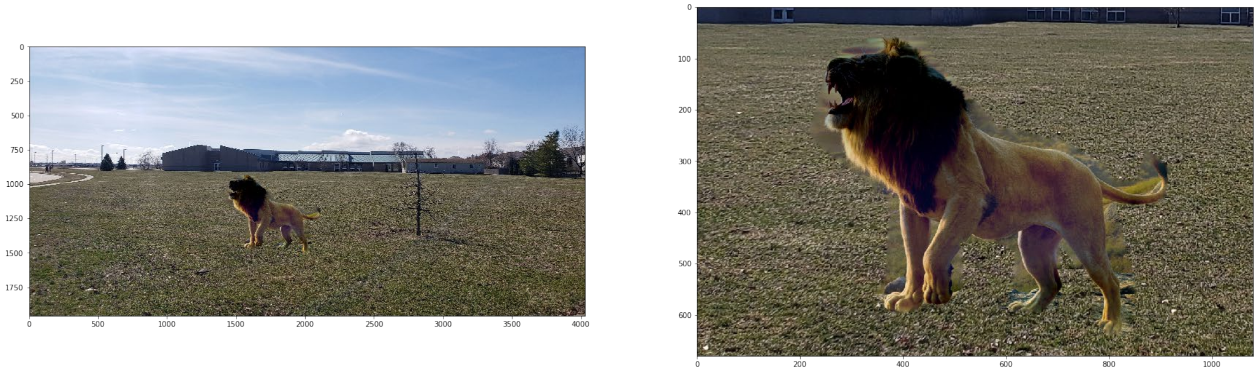
In [176]:

```
%matplotlib inline
import matplotlib.pyplot as plt
cropped_object, object_mask = align_source(object_img, mask, background_im
g, bottom_center)
```



In [177]:
```
output, output_mask = poisson_blend(cropped_object, object_mask, backgroun
d_img,mask)
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(30, 30)
axes[0].imshow(output)
axes[1].imshow(output_mask)
plt.savefig('lion_feild.jpg')
```

```
0
1
2
```



Part 3 Mixed Gradients (20 pts)

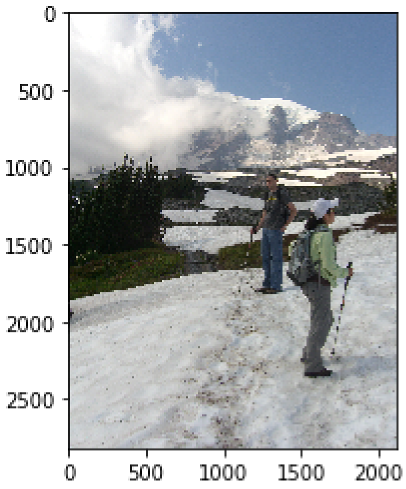
In [155]:
```
def mix_blend(cropped_object, object_mask, background_img, mask):
    """
    :param cropped_object: numpy.ndarray One you get from align_source
    :param object_mask: numpy.ndarray One you get from align_source
    :param background_img: numpy.ndarray
    """
    #TO DO
    row_start = int(bottom_center[1] - (mask.shape[0]))
    col_start = int(bottom_center[0] - (mask.shape[1]/2))
    pad = 40

    output = background_img


    background_img = background_img[row_start - pad:row_start + mask.shape
[0] + pad, col_start - pad: col_start + mask.shape [1] + pad]
    cropped_object = cropped_object[row_start - pad:row_start + mask.shape
[0] + pad, col_start - pad: col_start + mask.shape [1] + pad]
    object_mask = object_mask[row_start - pad:row_start + mask.shape[0] +
pad, col_start - pad: col_start + mask.shape [1] + pad]
    #return cropped_object





    #return cropped_object, background_img
    background_rows = background_img.shape[0]
    background_cols = background_img.shape[1]
    background_count = background_rows * background_cols

    output_mask = np.zeros(shape = (background_rows, background_cols,3))

    #print(background_count)
    im2var =  np.arange(background_count).reshape(background_rows, backgro
und_cols)

    #print(im2var)


    for z in range(3):
        v = []
        sparse_value = []
        sparse_row = []
        sparse_col = []
        b = []
        e = 0
        #A =  np.zeros(shape=(background_count,background_count))

        for y in range(background_rows):
            for x in range(background_cols):
                if not object_mask[y,x]: #background only
                    sparse_value.append(1)
                    sparse_row.append(e)
                    sparse_col.append(im2var[y,x])
                    b.append(background_img[y,x,z])
                    e = e + 1

                else:
                    if object_mask[y,x+1]:
                        sparse_value.append(-1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x + 1])

                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])

                        background_gradiant = abs(background_img[y,x,z] -
background_img[y,x+1,z])
                        source_gradiant = abs(cropped_object[y,x,z] - crop
ped_object[y,x+1,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_img[y,x+1,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_object[y,x+1,z])
                            e = e + 1

                    if object_mask[y+1,x]:
                        sparse_value.append(-1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y+1,x])

                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])


                        background_gradiant = abs(background_img[y,x,z] - background_img[y+1,x,z])
                        source_gradiant = abs(cropped_object[y,x,z] - cropped_object[y+1,x,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_img[y+1,x,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_object[y+1,x,z])
                            e = e + 1

                    if not object_mask[y,x+1]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])


                        background_gradiant = abs(background_img[y,x,z] - background_img[y,x+1,z])
                        source_gradiant = abs(cropped_object[y,x,z] - cropped_object[y,x+1,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_img[y,x+1,z] + background_img[y,x+1,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_object[y,x+1,z] + background_img[y,x+1,z])
                            e = e + 1

                    if not object_mask[y,x-1]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])
                        background_gradiant = abs(background_img[y,x,z] -
background_img[y,x-1,z])
                        source_gradiant = abs(cropped_object[y,x,z] - crop
ped_object[y,x-1,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_i
mg[y,x-1,z] + background_img[y,x-1,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_obje
ct[y,x-1,z] + background_img[y,x-1,z])
                            e = e + 1

                    if not object_mask[y+1,x]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])


                        background_gradiant = abs(background_img[y,x,z] -
background_img[y+1,x,z])
                        source_gradiant = abs(cropped_object[y,x,z] - crop
ped_object[y+1,x,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_i
mg[y+1,x,z] + background_img[y+1,x,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_obje
ct[y+1,x,z] + background_img[y+1,x,z])
                            e = e + 1

                    if not object_mask[y-1,x]:
                        sparse_value.append(1)
                        sparse_row.append(e)
                        sparse_col.append(im2var[y,x])


                        background_gradiant = abs(background_img[y,x,z] -
background_img[y-1,x,z])
                        source_gradiant = abs(cropped_object[y,x,z] - crop
ped_object[y-1,x,z])

                        if background_gradiant > source_gradiant:
                            b.append(background_img[y,x,z]  - background_i
mg[y-1,x,z] + background_img[y-1,x,z])
                            e = e + 1
                        else:
                            b.append(cropped_object[y,x,z]  - cropped_obje
ct[y-1,x,z] + background_img[y-1,x,z])
                            e = e + 1

        sparse_value = np.asarray(sparse_value)
        sparse_row = np.asarray(sparse_row)
        sparse_col = np.asarray(sparse_col)
        b = np.asarray(b).T
        #return 0
        A = csr_matrix((sparse_value, (sparse_row, sparse_col)), shape=(e,
 background_count))
        print(z)
        v = lsqr(A, b)

        send = np.clip(v[0], a_min = 0, a_max = 1)
        output[row_start - pad:row_start + mask.shape[0] + pad, col_start
- pad: col_start + mask.shape [1] + pad,z] = np.reshape(send, (background_
rows, background_cols))
        output_mask[:,:,z] = np.reshape(send, (background_rows, background
_cols))


    return output, output_mask









    #TO DO
    pass
```

In [120]:

```
# Feel free to change image
background_img = cv2.cvtColor(cv2.imread('samples/im2.JPG'), cv2.COLOR_BGR
2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```

Out[120]:

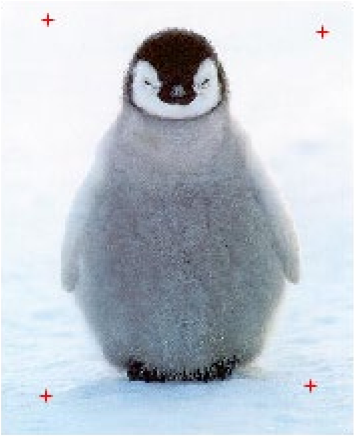
```
<matplotlib.image.AxesImage at 0x1bd90db6710>
```



In [121]:

```
# Feel free to change image
object_img = cv2.cvtColor(cv2.imread('samples/penguin-chick.jpeg'), cv2.CO
LOR_BGR2RGB).astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

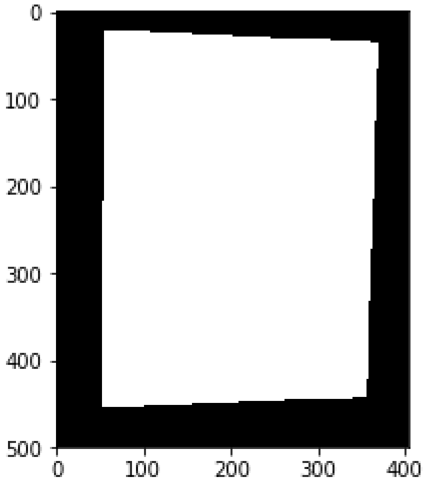
```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [122]:

```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

```
<matplotlib.figure.Figure at 0x1bd83c463c8>
```



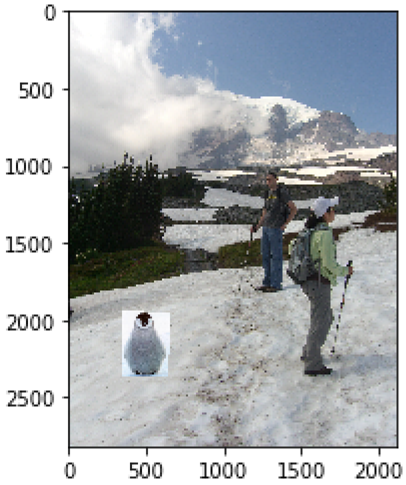
In [127]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
bottom_center = specify_bottom_center(background_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
 Also, make sure the object fill fit into the background image. Otherwise
```

```
it will crash
```

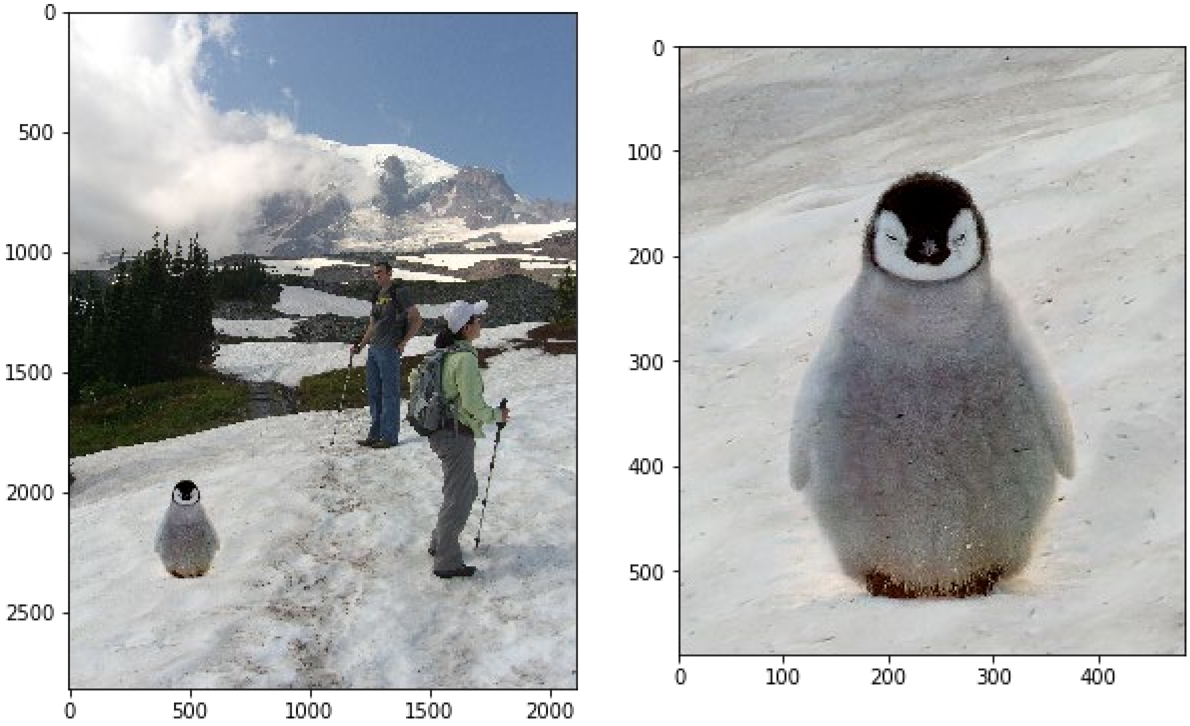
In [128]:
```
%matplotlib inline
import matplotlib.pyplot as plt
cropped_object, object_mask = align_source(object_img, mask, background_im
g, bottom_center)
```



In [129]:
```
output, output_mask = mix_blend(cropped_object, object_mask, background_im
g, mask)
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(10, 10)
axes[0].imshow(output)
axes[1].imshow(output_mask)
plt.savefig('penguin_mountain.jpg')
```

```
0
1
2
```

Out[129]:
```
<matplotlib.image.AxesImage at 0x1bd90ec4cc0>
```

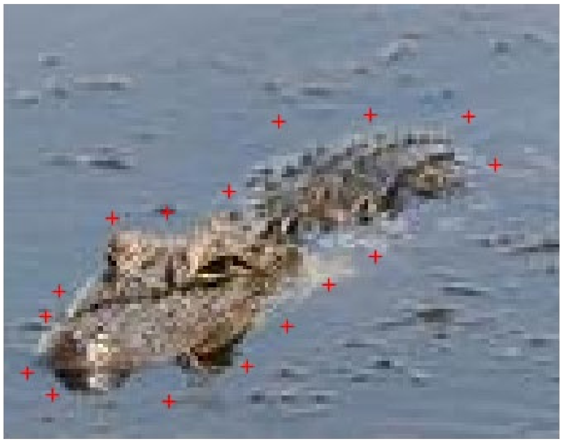


In [197]:
```
# Feel free to change image
background_img = cv2.cvtColor(cv2.imread('samples/pool.JPG'), cv2.COLOR_BG
R2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```

Out[197]:
```
<matplotlib.image.AxesImage at 0x2105432c208>
```

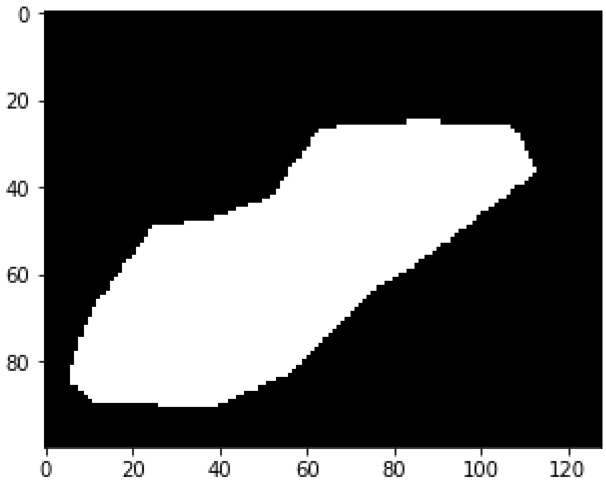
In [198]:
```
# Feel free to change image
object_img = cv2.cvtColor(cv2.imread('samples/alligator_small.jpg'), cv2.C
OLOR_BGR2RGB).astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again.
```



In [199]:
```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

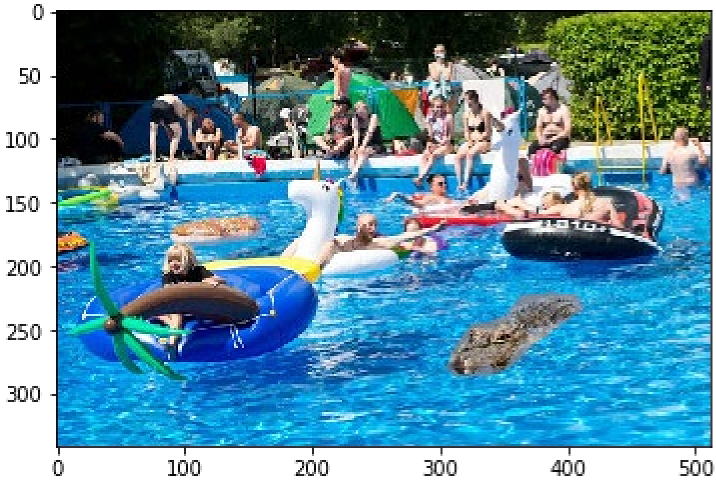
```
<matplotlib.figure.Figure at 0x2105b1f8080>
```



In [200]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
bottom_center = specify_bottom_center(background_img)
```

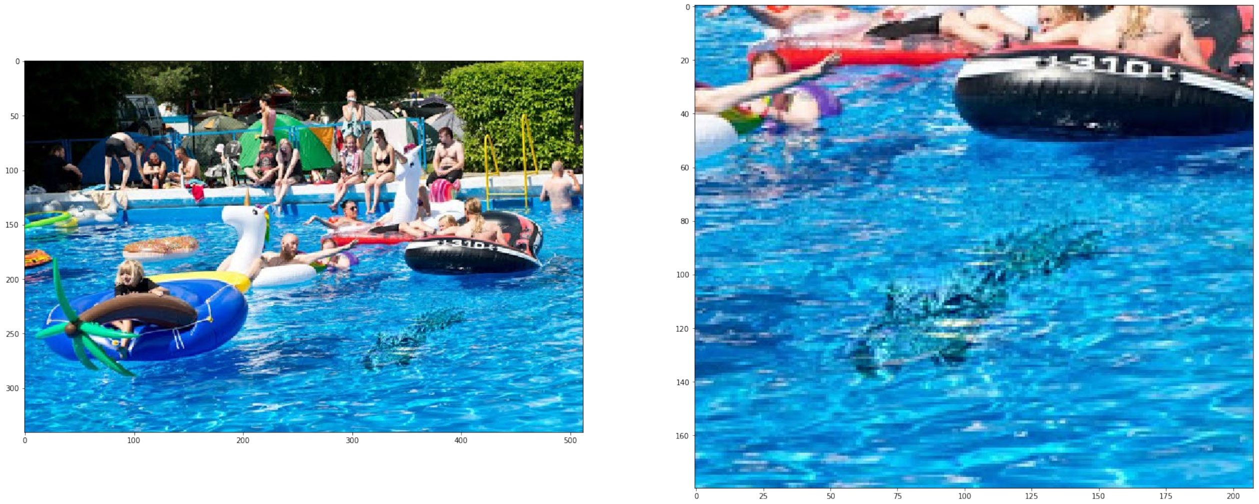
```
If it doesn't get you to the drawing mode, then rerun this function again.
 Also, make sure the object fill fit into the background image. Otherwise
it will crash
```

In [201]:
```
%matplotlib inline
import matplotlib.pyplot as plt
cropped_object, object_mask = align_source(object_img, mask, background_im
g, bottom_center)
```



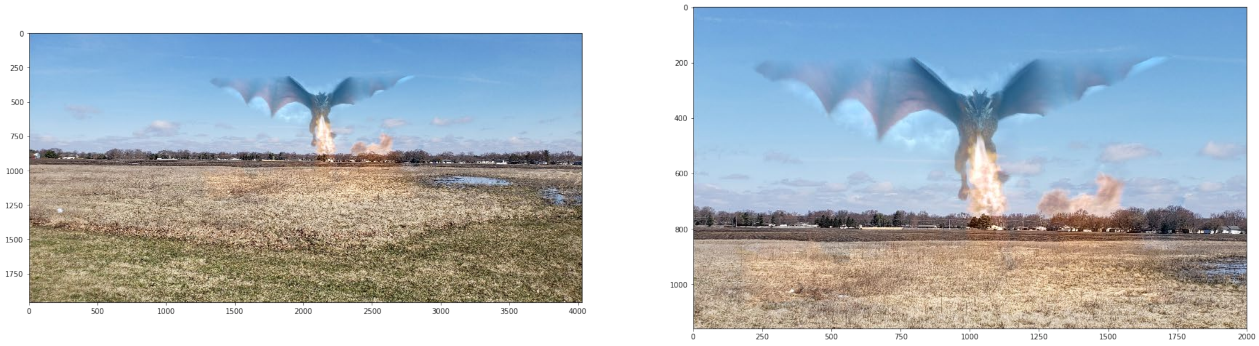
In [202]:
```
output, output_mask = mix_blend(cropped_object, object_mask, background_im
g, mask)
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(30, 30)
axes[0].imshow(output)
axes[1].imshow(output_mask)
plt.savefig('pool_aligator.jpg')
```

```
0
1
2
```



In [156]:
```
output, output_mask = mix_blend(cropped_object, object_mask, background_im
g,mask)
fig, axes = plt.subplots(1, 2)
fig.set_size_inches(30, 30)
axes[0].imshow(output)
axes[1].imshow(output_mask)
plt.savefig('dragon_feild_mix_blend.jpg')
```

```
0
1
2
```

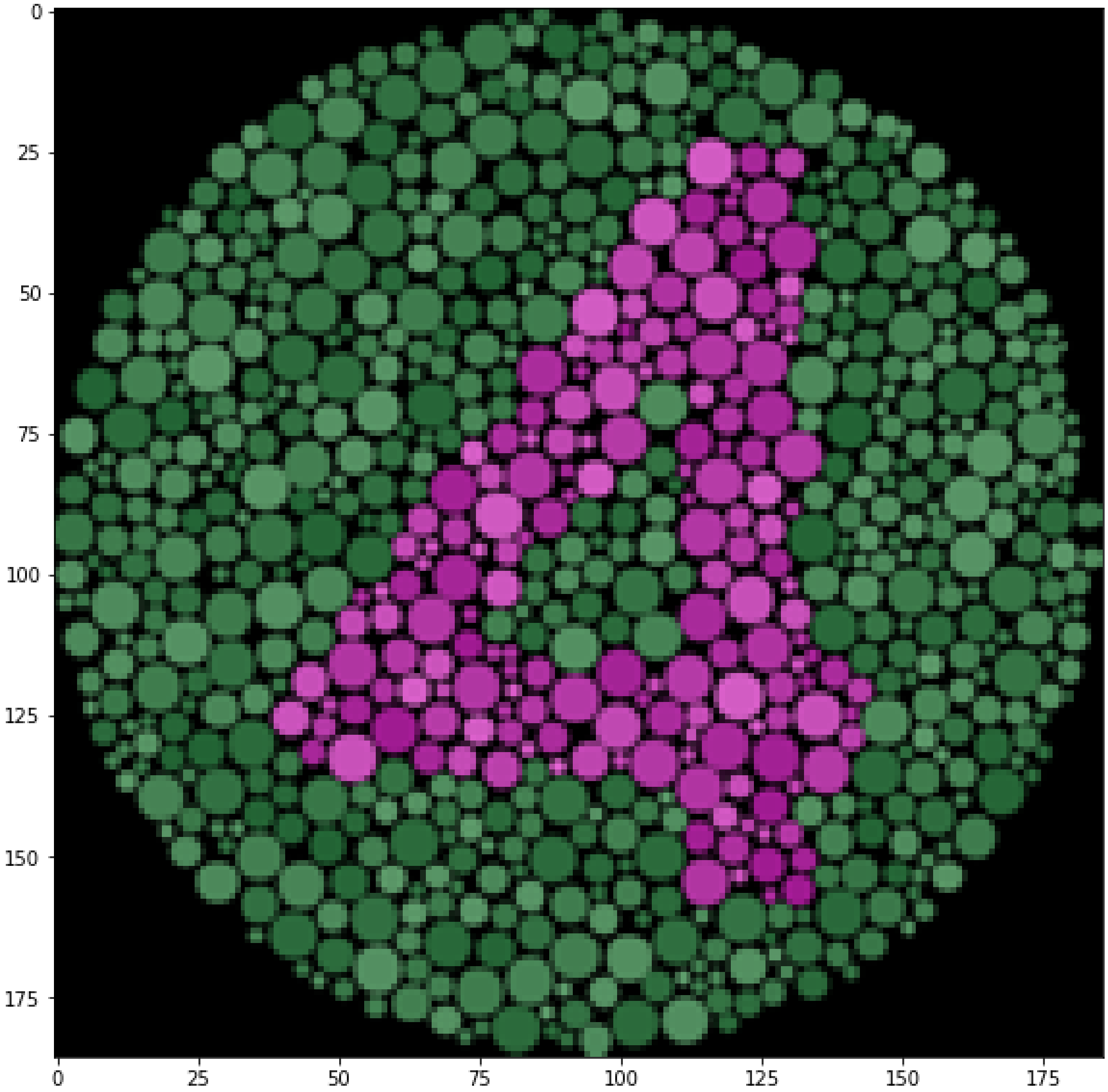


Bells & Whistles (Extra Points)

Color2Gray (20 pts)

In [5]:
```
img = cv2.cvtColor(cv2.imread('samples/colorBlind4.png'), cv2.COLOR_BGR2RG
B).astype('double') / 255.0
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(img)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x21042206d30>
```

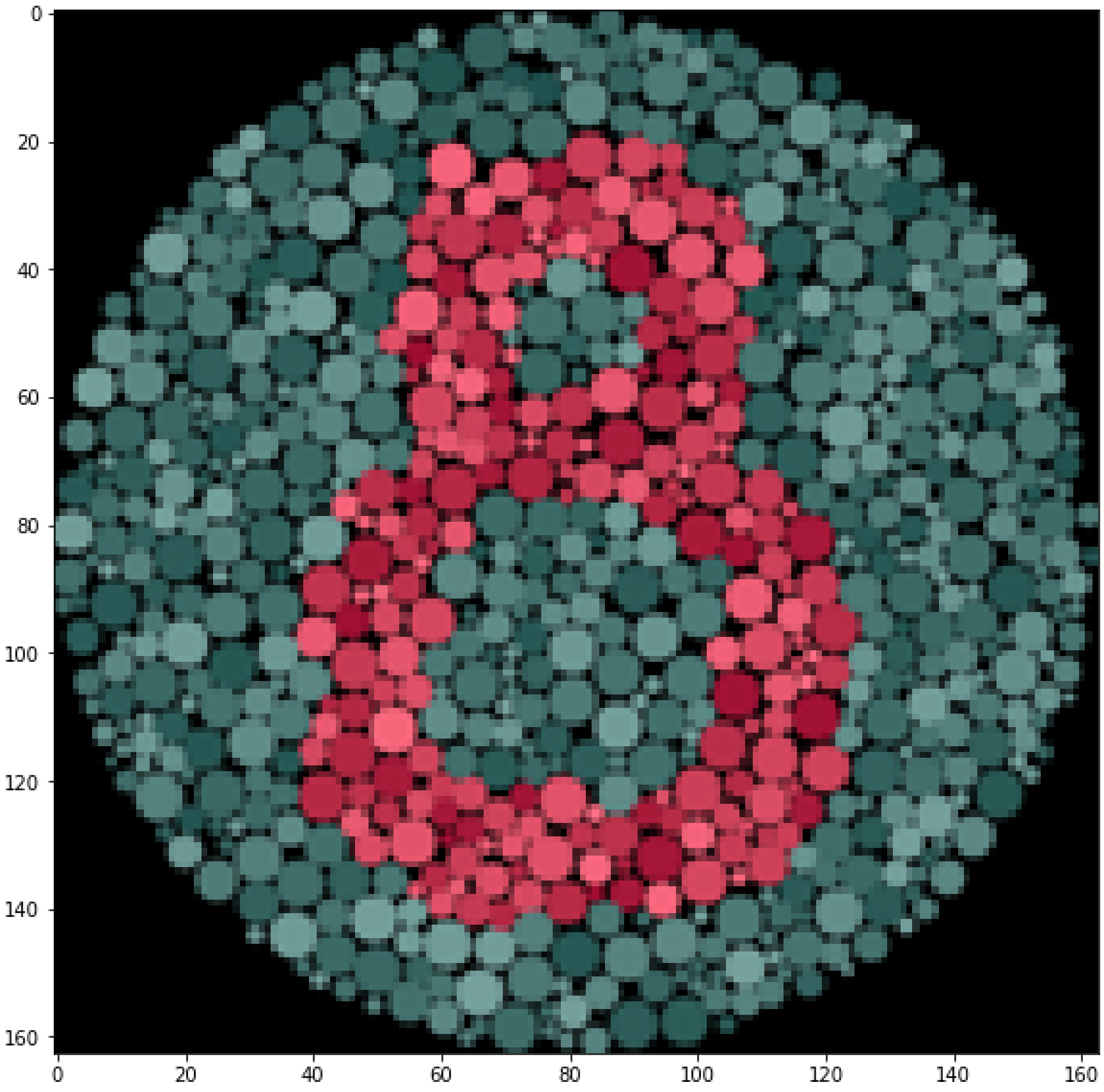


In [205]:

```
img = cv2.cvtColor(cv2.imread('samples/colorBlind8.png'), cv2.COLOR_BGR2RGB).astype('double') / 255.0
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(img)
```

Out[205]:

```
<matplotlib.image.AxesImage at 0x21046f5a278>
```

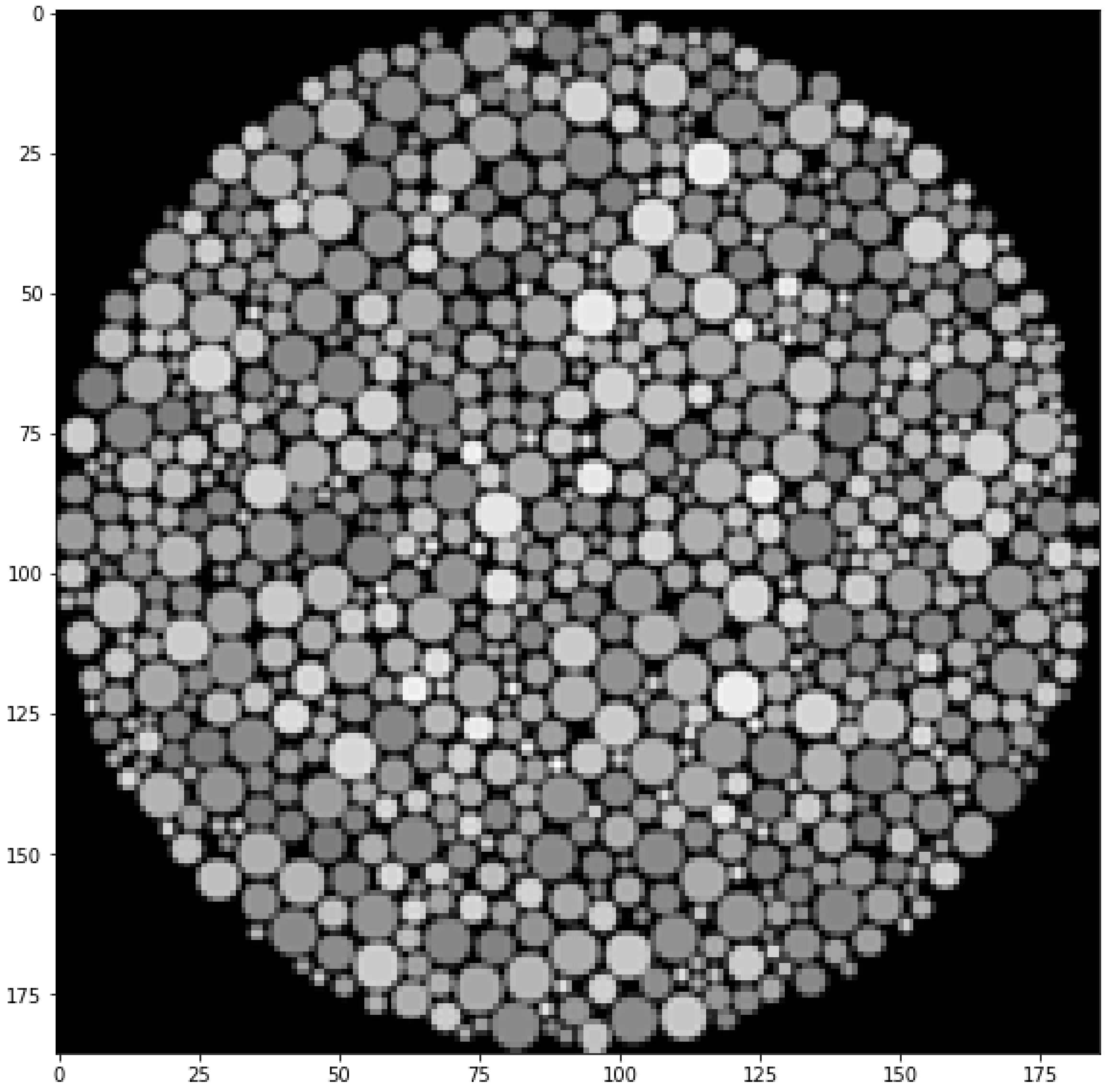


In [204]:

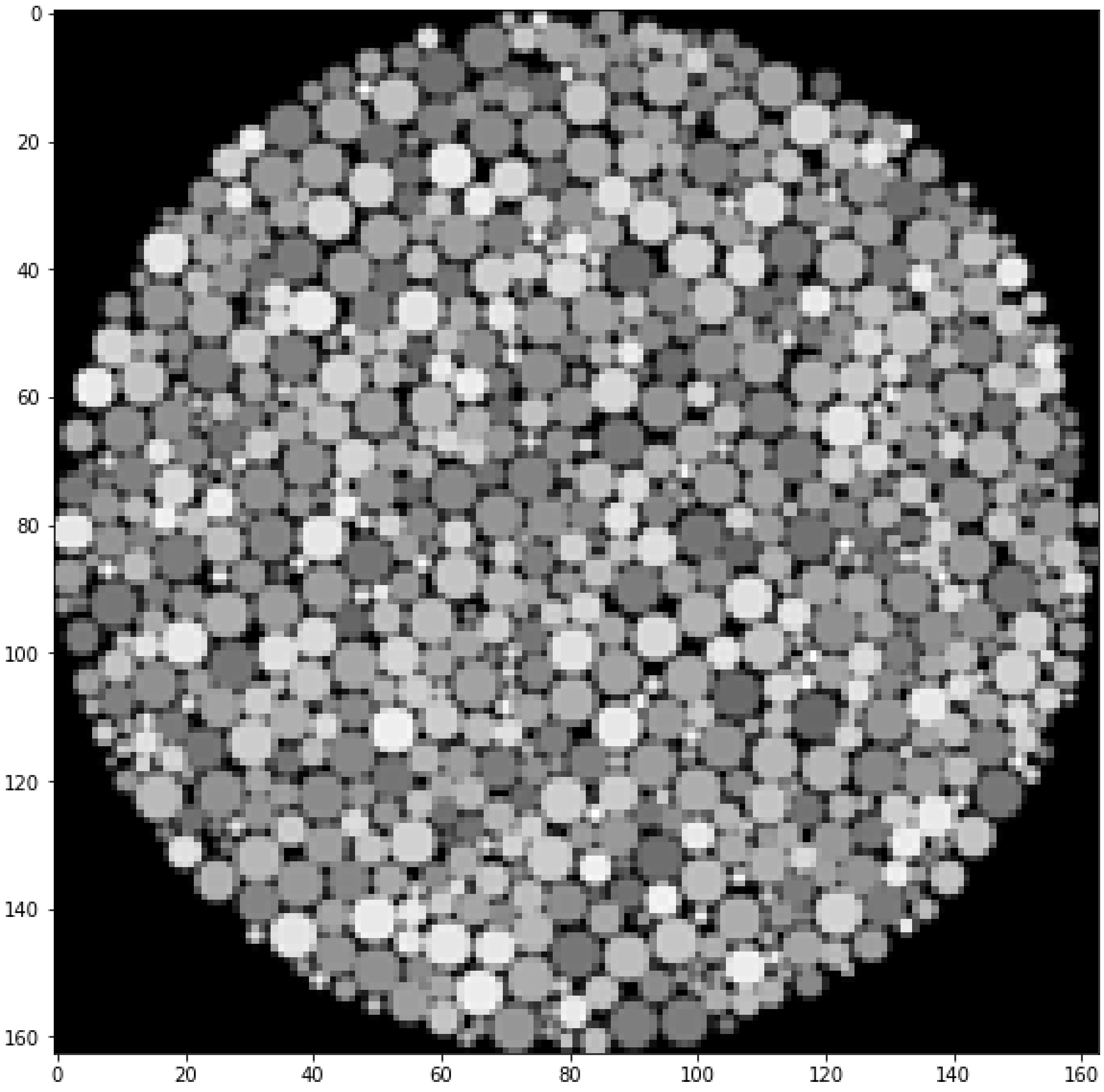
```
img = cv2.cvtColor(cv2.imread('samples/colorBlind4.png'), cv2.COLOR_BGR2GRAY).astype('double') / 255.0
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(img, cmap = 'gray')
```

Out[204]:

```
<matplotlib.image.AxesImage at 0x2105c050b00>
```



```
In [206]: img = cv2.cvtColor(cv2.imread('samples/colorBlind8.png'), cv2.COLOR_BGR2GR
          AY).astype('double') / 255.0
          fig, axes = plt.subplots(1, 1)
          fig.set_size_inches(10, 10)
          axes.imshow(img, cmap = 'gray')

Out[206]: <matplotlib.image.AxesImage at 0x21048a1dbe0>
```



In [84]:

```
def color2gray(img):


    color_image = cv2.cvtColor(cv2.imread(img), cv2.COLOR_BGR2RGB).astype(
'double') / 255.0
    gray_image = cv2.cvtColor(cv2.imread(img), cv2.COLOR_BGR2GRAY).astype(
'double') / 255.0


    rows = color_image.shape[0]
    cols = color_image.shape[1]
    im2var = np.arange(rows * cols).reshape(rows, cols)
    output = np.zeros(shape = (rows, cols))


    sparse_value = []
    sparse_row = []
    sparse_col = []
    b = []
    e = 0
    for y in range(0,rows):
        for x in range(cols - 1):
            sparse_value.append(1)
            sparse_row.append(e)
            sparse_col.append(im2var[y,x+1])

            sparse_value.append(-1)
            sparse_row.append(e)
            sparse_col.append(im2var[y,x])

            value_max_1 = (color_image[x,y,0])
            value_max_2 = (color_image[x+1,y,0])
            for z in range(1,3):
                if value_max_1 < color_image[x,y,z]:
                    value_max_1 = (color_image[x,y,z])
                if value_max_2 < color_image[x+1,y,z]:
                    value_max_2 = (color_image[x+1,y,z])


            b.append(value_max_2 - value_max_1)
            e = e + 1


    for y in range(0,rows -1):
        for x in range(0, cols):
            sparse_value.append(1)
            sparse_row.append(e)
            sparse_col.append(im2var[y+1,x])

            sparse_value.append(-1)
            sparse_row.append(e)
            sparse_col.append(im2var[y,x])



            value_max_1 = (color_image[x,y,0])
            value_max_2 = (color_image[x,y+1,0])
            for z in range(1,3):
                if value_max_1 < color_image[x,y,z]:
                    value_max_1 = (color_image[x,y,z])
                if value_max_2 < color_image[x,y+1,z]:
                    value_max_2 = (color_image[x,y+1,z])

            b.append(value_max_2 - value_max_1)
            e = e + 1

    sparse_value.append(1)
    sparse_row.append(e)
    sparse_col.append(im2var[0,0])
    b.append((color_image[x,y,0] + color_image[x,y,1] + color_image[x,y,2]
) / 3)
    print(len(sparse_value),len(sparse_row),len(sparse_col),len(b))
    print(rows*cols)
    print(e)
    sparse_value = np.asarray(sparse_value)
    sparse_row = np.asarray(sparse_row)
    sparse_col = np.asarray(sparse_col)
    b = np.asarray(b).T

    A = csr_matrix((sparse_value, (sparse_row, sparse_col)), shape=(e+1, r
ows*cols))
    v = lsqr(A, b)
    send = np.clip(v[0], a_min = 0, a_max = 1)

    output = np.reshape(send, (rows, cols))
    output = np.rot90(output)
    output = cv2.flip(output, 0)
    return output

    pass
```

In [85]:

```
img = 'samples/colorBlind4.png'
```

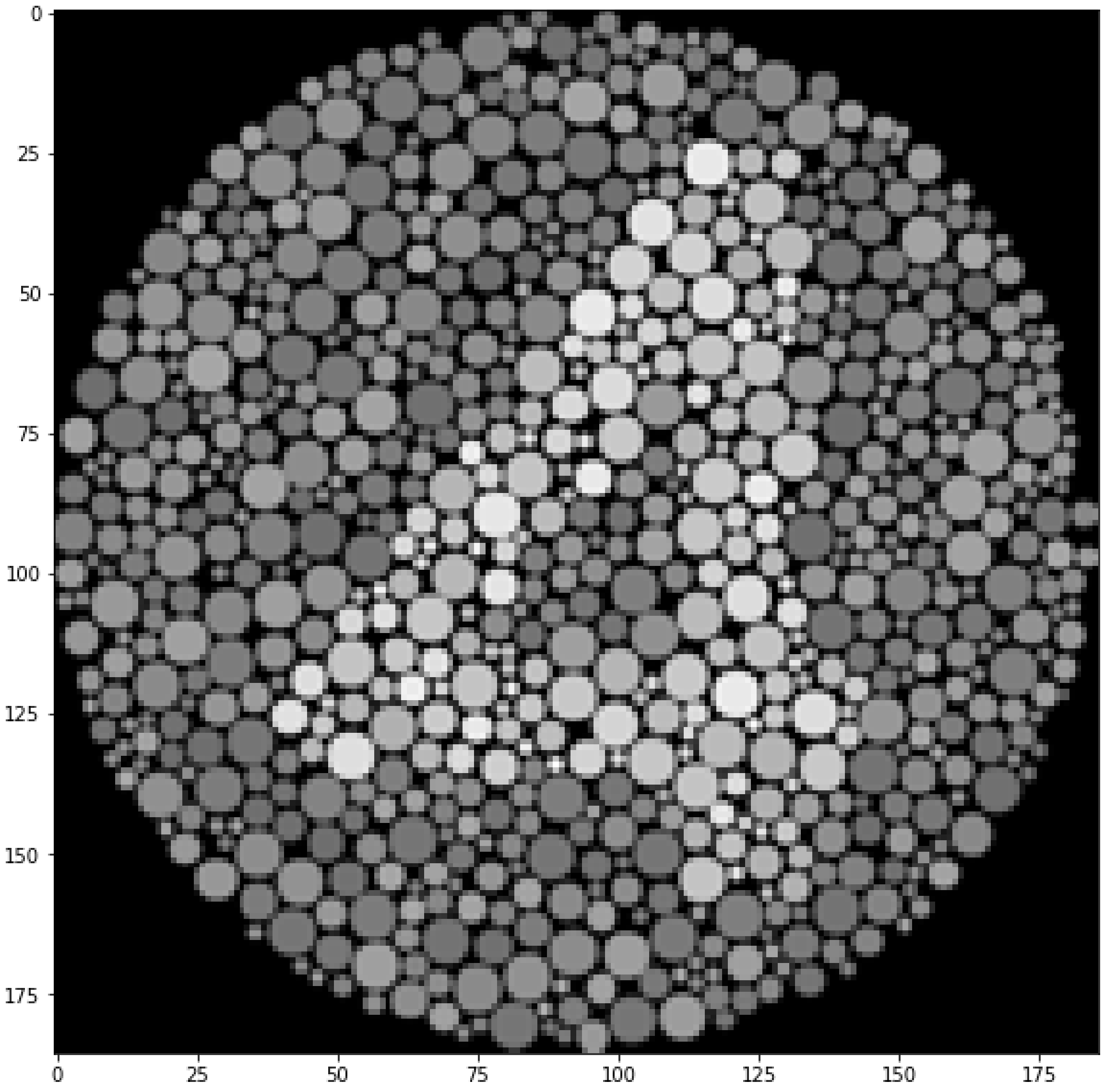
In [86]:

```
gray_image = color2gray(img)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(gray_image, CMAP='gray')
```

```
137641 137641 137641 68821
34596
68820
```

Out[86]:

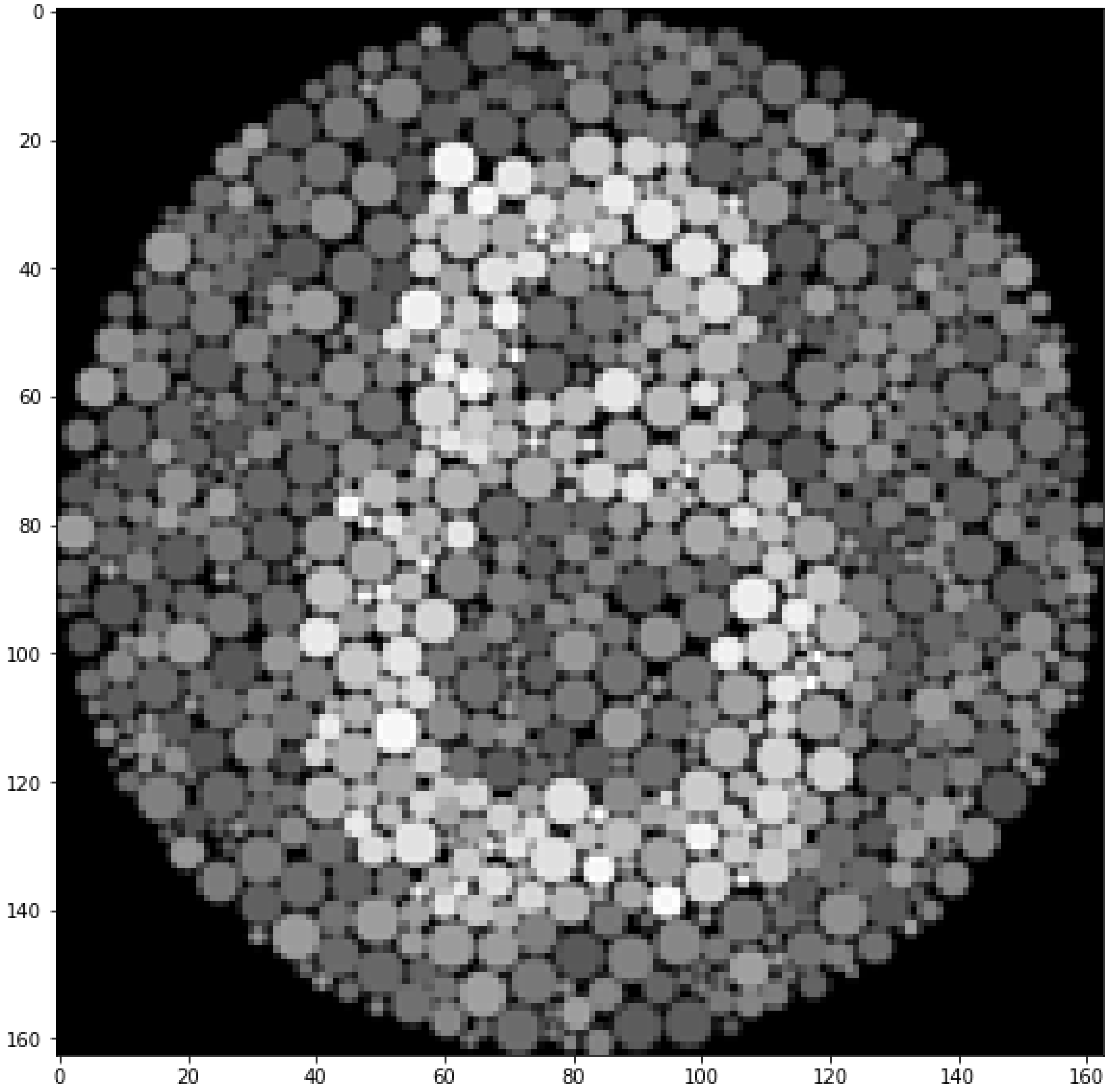
```
<matplotlib.image.AxesImage at 0x210448d1dd8>
```



In [87]:
```
img_2 = 'samples/colorBlind8.png'
gray_image = color2gray(img_2)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(gray_image, CMAP='gray')
```

```
105625 105625 105625 52813
26569
52812
```

Out[87]:
```
<matplotlib.image.AxesImage at 0x21046fa3cc0>
```



Laplacian pyramid blending (20 pts)

In [50]:

```python
def laplacian_blend(img1, img2, pyramid_height):

    cols = img1.shape[1]
    img1_img2 = np.hstack((img1[:, :int(cols/2)], img2[:, int(cols/2):]))

    img1_g = []
    img1_lp  = []
    img2_g = []
    img2_lp = []
    total = []


    img1_g.append(img1)
    img2_g.append(img2)
    for x in range(pyramid_height):
        img1 = cv2.pyrDown(img1)
        img2 = cv2.pyrDown(img2)
        img1_g.append(img1)
        img2_g.append(img2)

    img1_lp.append(img1_g[pyramid_height - 1])
    img2_lp.append(img2_g[pyramid_height - 1])

    for x in range(pyramid_height - 1,0,-1):
        lp1 = img1_g[x-1] - cv2.pyrUp(img1_g[x])
        img1_lp.append(lp1)
        lp2 = img2_g[x-1] - cv2.pyrUp(img2_g[x])
        img2_lp.append(lp2)

    for lp1, lp2 in zip(img1_lp, img2_lp):
        cols = lp1.shape[1]
        lp = np.hstack((lp1[:, 0:int(cols/2)], lp2[:, int(cols/2):]))
        total.append(lp)

    output = total[0]
    for x in range(1,pyramid_height):
        output = cv2.pyrUp(output)
        output = total[x] + output


    output = np.clip(output, a_min = 0, a_max = 1)
    return output, img1_img2
```

In [60]:

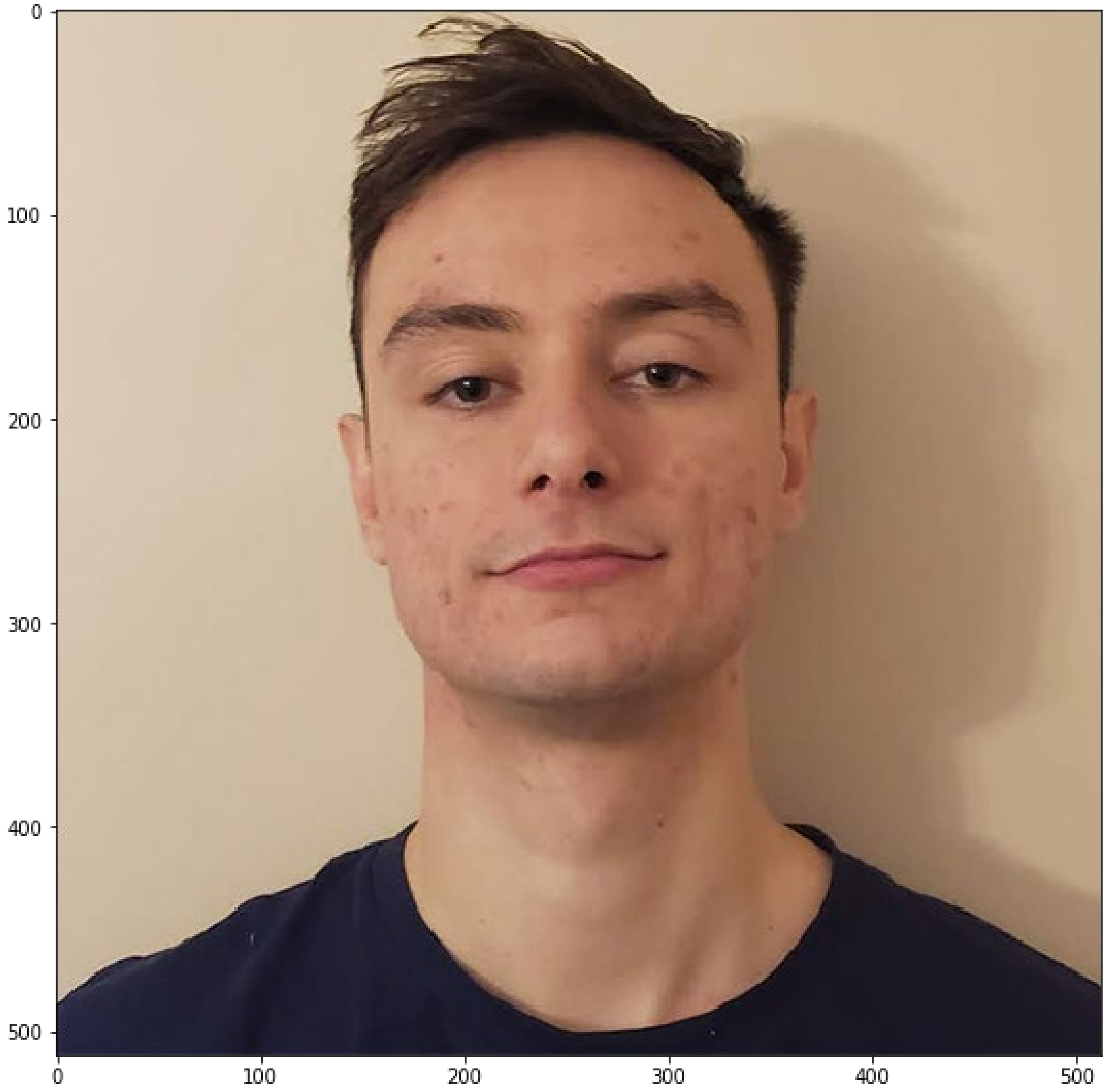
```
tarik = cv2.cvtColor(cv2.imread('samples/tarik.jpg'), cv2.COLOR_BGR2RGB).a
stype('double') / 255.0
amar = cv2.cvtColor(cv2.imread('samples/amar.jpg'), cv2.COLOR_BGR2RGB).ast
ype('double') / 255.0
```

In [61]:

```
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(tarik)
```

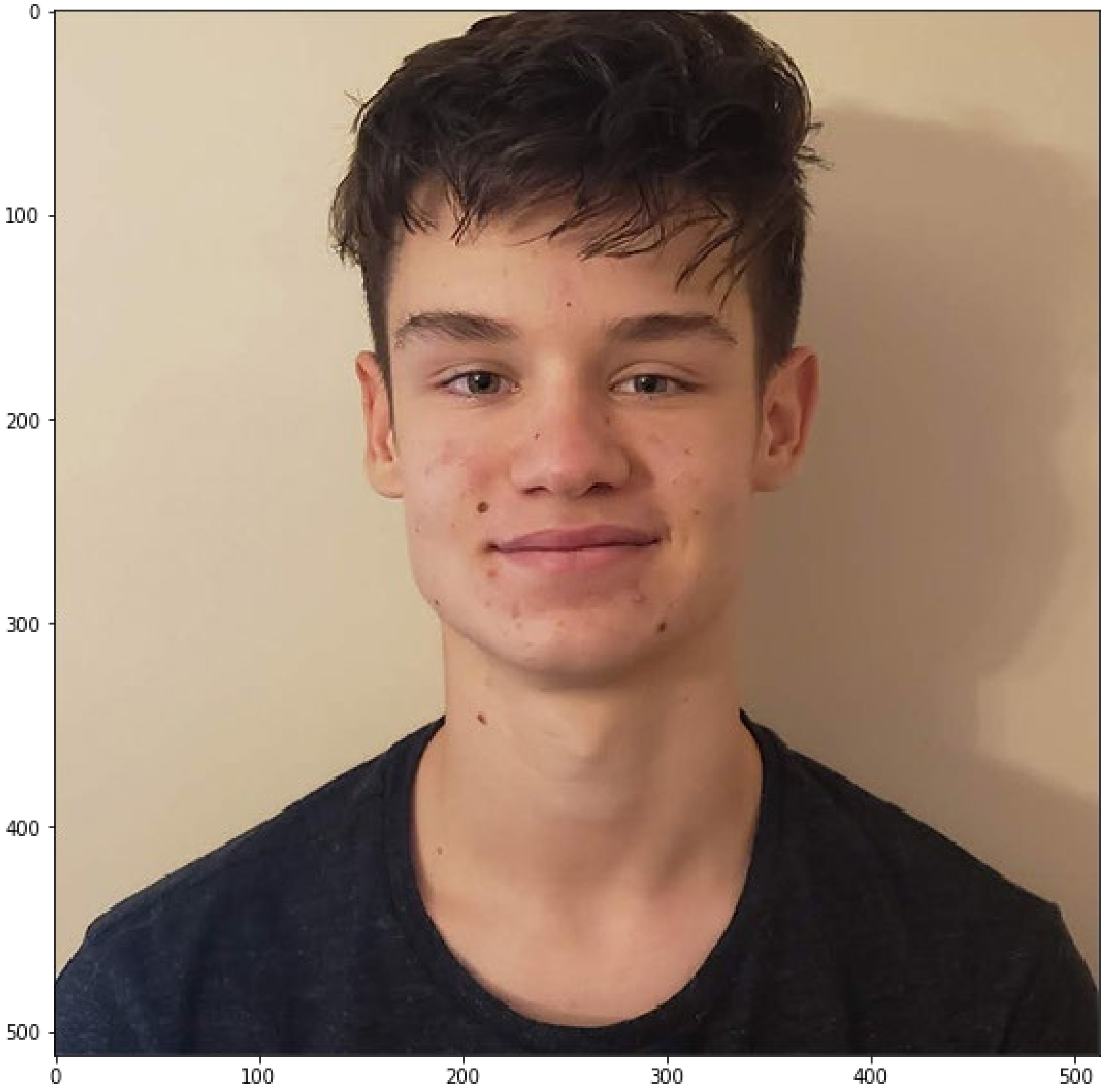
Out[61]:

```
<matplotlib.image.AxesImage at 0x29e86496da0>
```



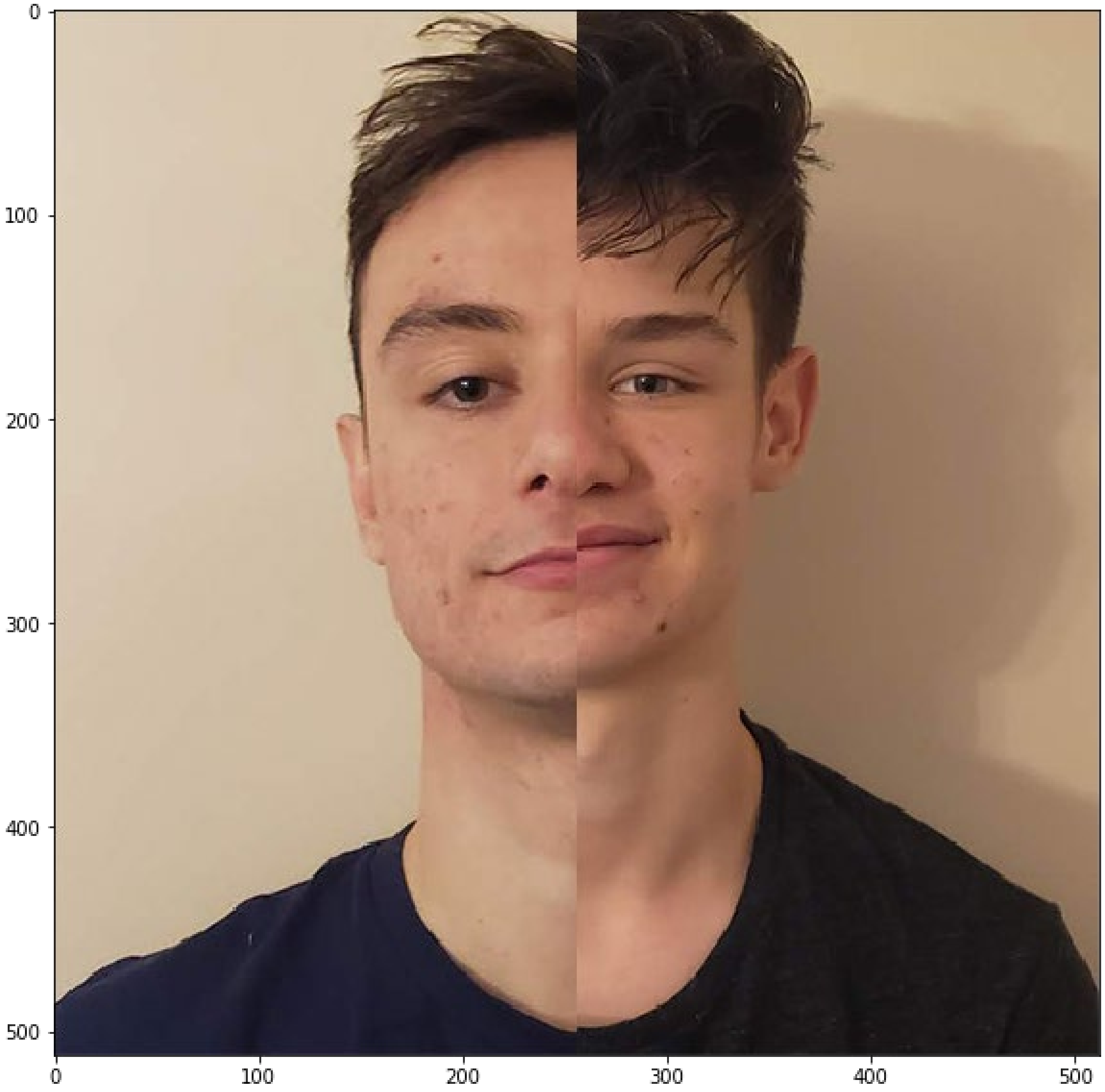
In [62]:
```
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(amar)
```

Out[62]:
```
<matplotlib.image.AxesImage at 0x29e8a0965f8>
```



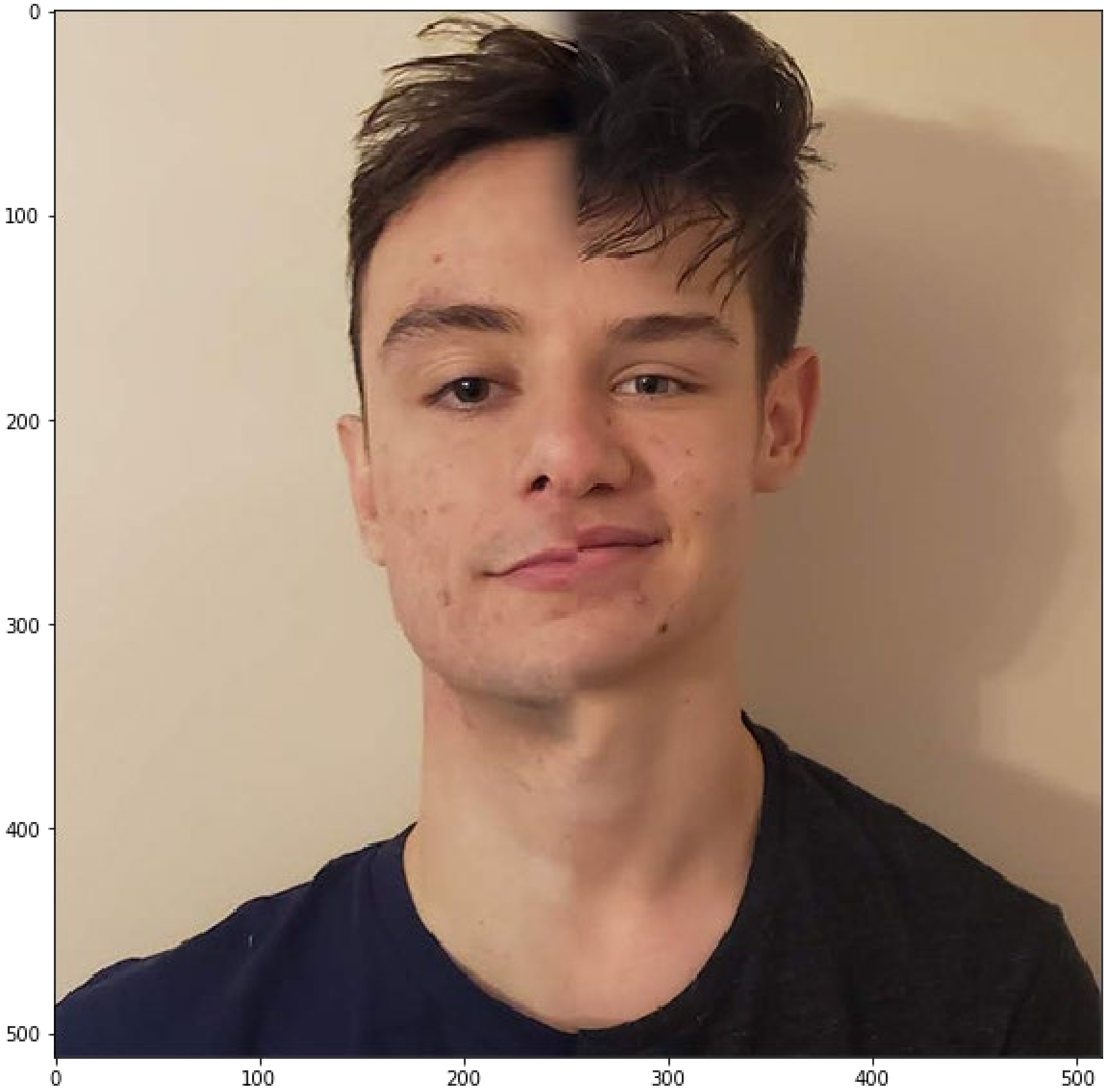
In [58]:
```
output, non_blend_output = laplacian_blend(tarik, amar,5)

fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(non_blend_output)
```

Out[58]:
```
<matplotlib.image.AxesImage at 0x29e86686828>
```



```
In [59]: fig, axes = plt.subplots(1, 1)
         fig.set_size_inches(10, 10)
         axes.imshow(output)
```

```
Out[59]: <matplotlib.image.AxesImage at 0x29e86370eb8>
```



More gradient domain processing (up to 20 pts)